Page Flow Overview

Table of contents

1. Introduction

A Java Page Flow (**JPF**) is a group of pages (often JavaServer Pages - **JSPs**) and an annotated Java "controller" class that defines the flow of an application. In addition to allowing complex logic to dictate which pages are displayed, JPF technology helps decouple page authoring from application logic by preventing one page from directly referencing another. It also allows for easy state management, and reuse of entire flows within other flows.

2. Page Flow Features

Page Flows make building Java web applications easy and intuitive. When programming with Page Flows, the developer writes Java classes and pages --that's it. There is very little occasion to work with configuration files, or other components. Page Flow programming also excels at separating presentation logic from data processing logic, resulting in uncluttered JSP code which is easy to understand and edit. Data processing and the web application configurables are handled in a single Java class using a simple declarative programming model.

Declarative Programming

Many common web app programming tasks are accomplished through a declarative programming model using JSR 175 metadata annotations, a new feature in JKD5. JSR 175 metadata annotations, "annotations" for short, are property setters for Java classes and methods, alleviating the need for independent configuration files. Navigation, exception handling, validation, and other tasks become configurable properties of a single Java class, the "controller" class that drives the web application.

Page Flows are Stateful

When a user enters a Page Flow (by calling an URL in the Page Flow's URL space), an instance of the Page Flow's controller class is created. While the user is in the Page Flow, the controller class stores the accumulated flow-related state in member variables of the controller class. The methods within the class have access to the accumulated state, making for easy state management within the web application. For example, suppose your web application calls for a multi-page registration, where the user moves from page to page filling out a user profile. The controller class stores the user data as the user progresses through the registration and has access to the profile data as session state. When the user leaves the Page Flow, the state is automatically cleaned up.

Page Flows are Modular

A single web application can have multiple Page Flows within it, allowing you to break up the application into separate, self-contained chunks of functionality. For an example, see the Petstore Sample (../jpetstore.html) , which has different Page Flows for browsing the Petstore, buying products, and handling user accounts.

Page Flow web applications also contain a global Page Flow, called the "shared flow", which is both a fallback handler for unhandled actions and exceptions and a place to store session state. An instance of the shared flow class is stored in the user session upon the first request to any Page Flow and remains until the session ends. When an action is raised in a Page Flow, and that action is not handled by the Page Flow, the shared flow gets a chance to handle it. The same is true for an exception raised within a Page Flow: if it is unhandled in the Page Flow, the shared flow gets a chance to handle it.

Page Flows are Nestable

Page flow nesting gives you an even greater ability to break up your project into separate, self-contained bits of functionality. At its heart, it is a way of pushing aside the current page flow temporarily and transferring control to another (nested) page flow with the intention of coming back to the original (nesting) one.

So when would you use this? Nesting is useful when you want to do one of the following tasks:

- gather data from the user, for use in the current page flow
- allow the user to correct errors or supply additional information en route to executing a desired action
- show an alternate view of data represented in the current page flow
- show the user information that will be useful in the current page flow (e.g., help screens can be easily implemented as nested page flows)

Struts Integration

Page Flows are built on top of Apache Struts 1.1. Each Page Flow is compiled into a Struts module. As a result, Page Flow and Struts applications can work closely together.

Struts and Page Flow apps can co-habitate and interact with one another inside a web app. To forward from a Page Flow to a (pure) Struts module, simply reference the desired action within the Struts module. The same goes for the reverse direction: from a Struts module, simply configure an action to point to the desired method in the Page Flow.

You can also use the Struts merge feature to read configuration data from a pure Struts app into your Page Flow app's configuration files. Ordinarily, your Page Flow's configuration files are generated entirely from your application's JAVA source files (specifically from the *metadata annotations* that decorate the controller files). But, in cases where you want to

integrate a Struts module into your application, you can specify that the configuration files be generated from *both* the JAVA source files *and* the Struts module's configuration files, allowing you to change or add *any tag* in the generated configuration file. For example, suppose you want to override an action form's default scoping from request-scoping to session-scoping. To do this, you simply create a Struts configuration file that overrides the appropriate parts of the Page Flow's configuration file, and then refer to this override file from within the Page Flow's JAVA source file (= the controller file) using a special annotation. In particular, you would specify the override file to state that such-and-such an action form should have session-scope rather then request-scope (so that the action form can now be shared with the Struts app).

3. The Logical Flow

Writing traditional web applications without a JPF controller class requires a fair amount of logic to be applied within the application's pages. For example, a site that provides a "My Page" functionality for logged in users would have to include logic on the home page to determine if the "My Page" link should take the user to the login form or directly to their customized page.

Using a JPF, the home page of the application would not link directly to either the login page **or** the user's "My Page" location, but rather would point back into Java code that makes the decision.

For the rest of this overview, the following **logical page flow** will be used:

logical page flow

This flow supports several routes from the home page of the application to the user's "My Page":

1. The user may directly navigate from `index.jsp` to `mypage.jsp` (by clicking a link), if the user is already logged in.
2. If the user is not already logged in, attempts to navigate from `index.jsp` to `mypage.jsp` will be intercepted and the user will be taken to the `login.jsp` instead. After successfully logging in, the user will be automatically taken to `mypage.jsp`
3. The user may directly navigate from `index.jsp` to `login.jsp` (by clicking a link). After logging in, the user will be automatically taken to `mypage.jsp`.

   In the event of a login failure, `login.jsp` will be redisplayed to give them another opportunity to authenticate themselves.
4. If the user desires to register with the site, he can click a link that will take him to `signup.jsp`. One signed up, the `thanks.jsp` will be displayed which offers a link

to the `login.jsp` page.

4. The Implementation of the Flow: Controllers and Actions

In the above **logical flow** there are several *if* statements that cause the user flow to vary depending on their previous actions and other state.

- *If the user is not logged in...*
- *If the user is logged in...*
- *If the user's login attempt fails...*

Java Page Flows moves this condition logic out of the JSP pages and into a Java class that controls the movement through the application. This Java class is the **controller** portion of the **Model-View-Controller** (MVC) pattern. This allows a page to be written, for example, that appears to link directly from the home page of the application to the user's "My Page". The controller is given the opportunity to intercept the navigation between the two and redirect the user to the login page, if required.

Each of the interception points is an **action** of the particular controller class. Actions perform common application tasks. Here are some of the things that an action can do:

- navigate the user to a specified JSP
- perform conditional logic
- handle submitted data
- validate submitted data
- handle exceptions that arise in the application

Note that controller classes, and the actions they contain, are **URL addressable**. Hitting the following URL creates an instance of the controller class `foo.MyControllerClass` and runs its `begin` action. (When no other action is specified, the `begin` method is run by default.)

```
http://some.domain.com/foo/MyControllerClass.java
```

Hitting the following URL creates an instance of `foo.MyControllerClass` (if it doesn't already exist) and invokes the `someAction` action. Note that the controller class isn't mentioned by name: it's assumed that only one controller class exists in the directory, so there is only one candidate controller class to instantiate.

```
http://some.domain.com/foo/someAction.do
```

Note:

When a folder-path URL is called
http://some.domain.com/foo
the Controller class will not be instantiated, unless (1) you list the class in `web.xml`'s `<welcome-file-list>`.

```
<welcome-file-list>
  <welcome-file>Controller.java</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

> and (2) the Controller.java file resides in the folder `foo`. If the Controller.java resides in the `WEB-INF/src/` directory, then it will not be instantiated when the folder-path URL is called.

Actions may perform any required complex logic. For example, if a user clicks on the "My Page" link, the action may check if the user is logged in, and if so, navigate the user to the `mypage.jsp` page; otherwise it will navigate the user to the `login.jsp` page.

With normal HTML pages, each page is linked directly to other pages.

- **page > page > page > page**

When using JPFs, pages and actions are interwoven, transparently.

- **page > action > page > action > page > action > page**

The above **logical page flow** can be redrawn with JPF controller actions in mind, as:

implementation page flow

Now it is apparent that to navigate from `index.jsp` to `mypage.jsp`, the user traverses across the `myPage` action. This action performs the necessary check to determine if the user has already been authenticated. If the user has logged in already, it will direct the user straight to `mypage.jsp`; otherwise it will direct the user to `login.jsp`.

5. Next...

Next, learn about writing a **controller** class with actions.

- Page Flow Controller (pageflow_controllers.html)